Features of Event-Driven Message Queuing Architectures in Manufacturing: A Reference Model for Comparison

Stefan Gudenkauf [1], Javier Franke [2] and Janek Behrens [3]

Abstract: With increasing digitization and the use of cost-effective, ever more intelligent sensor and actuator systems at the edges of classic IT networks (edge computing), ever-increasing amounts of data are continuously being generated and sent from a wide variety of data sources. At the same time, manufacturing processes are subject to dynamic adjustments such as staff shortages, material shortages and fluctuations in energy costs. In addition, there is the requirement to demonstrate the sustainability of produced goods within the supply chain and to the outside world, which requires the collection of key figures across all levels of the manufacturing pyramid – and ideally also across the entire life cycle of the product. What they all have in common is that a very large amount of continuously and simultaneously operating systems can process sent data quickly and can react to relevant events in near real time. To build these systems, increasingly distributed applications based on the principle of message queuing (MQ) and event handling are attracting increasing interest in the manufacturing industry. In this paper[4], we surveyed recent models and architectures for such event-driven systems. Based on this survey, we propose a consolidated feature model to uniformly describe and evaluate event-driven manufacturing systems, regardless of whether an organization's own architectural needs or the offers of external providers are evaluated.

Keywords: Event-driven Architectures, Manufacturing Systems, Message Queuing, Feature Modeling.

1 Introduction

The automation systems established in the manufacturing industry are often a reflection of hierarchically organized processes and decision-making structures: they are hierarchically organized and communication is often *point-to-point*, see Fig. 1. Disadvantages of this hierarchy are the high communication effort and the limited flexibility due to rigid interfaces. In the context of current trends such as

[1] Jade University of Applied Sciences, 26389 Wilhelmshaven, Germany, stefan.gudenkauf@jade-hs.de, https://orcid.org/0000-0002-1813-3448
[2] Jade University of Applied Sciences, 26389 Wilhelmshaven, Germany, javier.franke@ jade-hs.de, https://orcid.org/0009-0005-0413-7076
[3] Jade University of Applied Sciences, 26389 Wilhelmshaven, Germany, janek.behrens@student.jade-hs.de, https://orcid.org/0000-0002-4825-1752
[4] This work is supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant No. 01MD22001B as part of the "Edge data economy" technology programme.

- **Internet of Things and Services (IoTS) and Industry 4.0** [SG16], in which parts of the automation hierarchy are regarded as cyber-physical production systems (CPPS),
- **Increasing service orientation** of the individual automation functions through cloud services and open service-oriented standards such as OPC UA [Di21], as well as
- **The need for ever-increasing optimization** of automation as a whole through methods of artificial intelligence (AI) and machine learning (ML) [Fr21],

these disadvantages are increasingly understood as competitive disadvantages. Therefore, manufacturing companies are striving to dissolve the classic automation hierarchy in favour of a service-oriented architecture whose services have to be defined and orchestrated. This requires suitable data-intensive applications [Kl17] that can record, store and distribute data across all levels of the automation hierarchy.

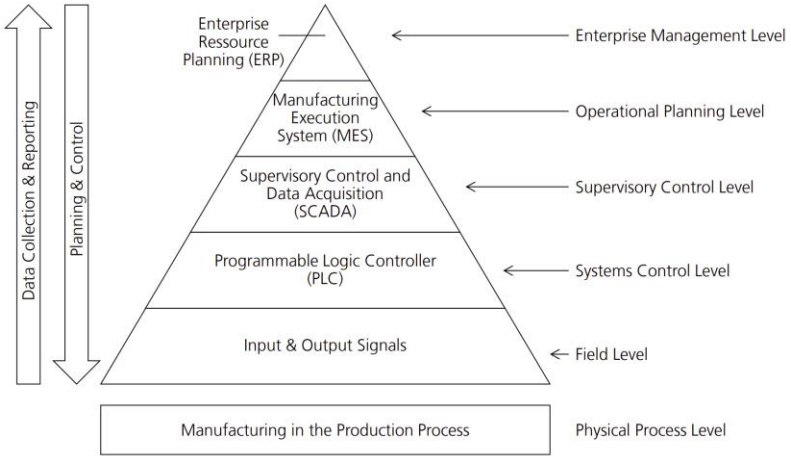


Fig. 1: Example of an automation hierarchy

1.1 Event-driven Architectures in the Manufacturing Industry

Traditional distributed applications such as distributed file systems such as HDFS [Bo08], cloud storage such as Amazon S3 [Pa08], and key value stores such as Apache Cassandra [LM10] are usually not sufficient for near real-time processing of such data streams [SJ14], [Fu21]. Instead, distributed applications based on the principle of message queuing (MQ)[5] are considered suitable [Fu21], [So18], [Ja18], [Ja20], [Yo19]. Common examples include Apache Kafka [Kr11], [Sh22], Apache Pulsar [Ra19], [Ta23], [Jo21], RabbitMQ [MS19], RocketMQ [Yo19], ZeroMQ [Ze23] and several

[5] Also often denoted as message-oriented middleware (MOM).

systems based on the Message Queuing Telemetry Transport (MQTT) protocol [MK20], [Mi18]. These technologies are currently being used successfully in software development in the context of microservices architectures [LF22], [Wo18] and the domain-driven design approach [Ev04]. Conversely, the use of microservices is also attracting increasing interest in the manufacturing industry, as individual services with a clearly defined function (single task) are being operated as separately distributable units (containers) – ideally as independently as possible of the underlying hardware infrastructure [Ce21]. Fig. 2 illustrates the commonalities and differences of such systems to traditional approaches on data processing.

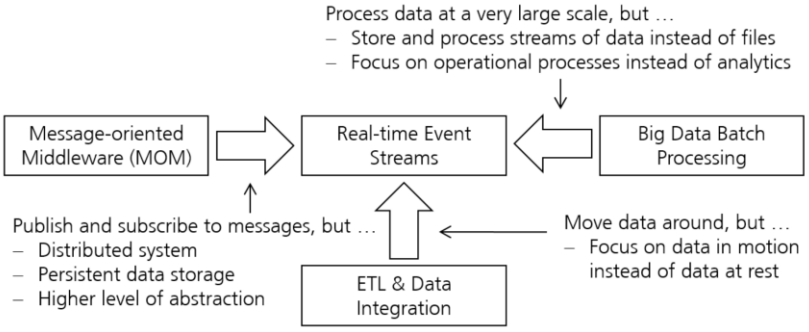


Fig. 2: Relations of broker architectures to other data processing approaches, cf. [Sh22]

The ongoing trend towards service orientation of the entire automation hierarchy now generates the opportunity to transfer the advantages of such event-based software architectures developed in operational software technology to a holistic, data-stream-based production in Industry 4.0 [SG16]. From the viewpoint of the data-providing and consuming services in production, these architectures then play the role of event-based integration platforms or simply as IoT brokers. However, potential users of such systems must find a way to uniformly describe and evaluate event-driven manufacturing systems, regardless of whether they want to formulate their own architectural needs or to evaluate the product offers of external providers.

The contribution of this paper is two-fold: (1) a survey of recent models and systems in manufacturing based on the notion of event-driven architecture (EDA) and in particular on message queuing (MQ), and (2) a consolidated feature model to uniformly describe and evaluate event-driven manufacturing systems, regardless of whether an organization's own architectural needs or the offers of external providers are evaluated. Section 2 presents the survey along with the description of the survey method, and Section 3 then discusses the features of event-driven MQ architectures in manufacturing. We discuss future work in Section 4 and summarize our findings in Section 5.

2 Subject of Research

According to the GQM approach of Basili et al. [Ba94] we formulate the goal of our work as the improvement (purpose) of the comparability (issue) of event-driven message-oriented middleware systems (object/process) from the viewpoint of the manufacturing industry (viewpoint). To do so, we examine the following questions:

1. What features do such event-driven systems of the recent past have?
2. Can we provide a reference model, with which one can describe and compare these systems?

To address the first question, this section presents an overview on several models and architectures of event-driven systems in the context of manufacturing, and a description of the survey method. We also summarize the differences between the surveyed approaches.

2.1 Survey Completeness

The works considered represent samples from academia and industry. We focused on approaches that (1) are based on the Event-Driven Architecture (EDA) software architecture paradigm and are (2) especially based on Message Queuing (MQ). Also (3), we considered approaches that are either generic or specifically oriented towards manufacturing/industrial production. Approaches that discuss specific aspects of MQ systems and architectures in-depth, such as performance and security, are excluded from the survey but may be added in future work. The considered approaches are:

- **Systems:** Apache Kafka [St18], Confluent Platform [Co20], Apache Pulsar [Ta23], RabbitMQ [MD19], LISA [Th15], Cybus Connectware [Cy22], and ZeroMQ [Pf22]
- **Models:** Bruns and Dunkel [BD10], Fu et al. [Fu21], Sommer et al. [So18], Yongguao et al. [Yo19], Dobbelaere and Esmaili [DE17], and Mishra and Kertesz [MK20]

2.2 Survey Method

The survey method is based on [Gu13] and is conducted in several steps. First, we began with a breadth-first search to identify relevant works. Although we did not apply a rigorous research method such as proposed by Kitchenham et al. [Ki07] [KB13], we used Google/Google Scholar, and SpringerLink as primary search platforms. Keywords included “event-driven”, “broker”, “message queuing” and “message-oriented” in conjunction with “manufacturing”. With the exception of Bruns and Dunkel [BD10], which we consider as a reference to EDA in general, the works considered were no older than 2015. We then selected works that comply with the selection criteria presented in

subsection 2.1. Next, we described the concepts discussed in these works using feature models [Ba05] [CE00] [CH06], since they combine accessibility, formality and structure discovery, as discussed by Gudenkauf et al. [Gu13]:

- **Accessibility:** "Similar to mind maps, feature models are simple and can be clearly arranged, represented, and compared to each other."
- **Formality:** "Feature models are formal models that can be validated. Also, they allow modeling exclusions, multiple selections, and cardinality."
- **Structure discovery:** "Feature models do not impose an organizational structure beforehand and even allow to model incomplete concepts. This is in contrast to tables, for example, in which columns and rows typically have to be named."

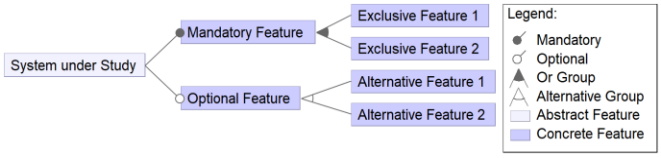


Fig. 3: An example of a feature model

Fig. 3 shows an example of a feature model. The root element represents a *system under study* (SUS). This element has two sub-features, of which one is mandatory (black-filled circle) and the other is optional (white-filled circle). Again, both features have sub-features. In general, the connection of a feature and its sub-features is described as an *and*-, *or*-, or *alternative*-group [Ba05] [Th14]. If a feature is selected, the selection of its parent feature is implied and all mandatory sub-features of an *and*-group must also be selected. An *or*-group implies that at least one sub-feature must be selected, and an *alternative*-groups implies that exactly one sub-feature has to be selected [Ba05] [Th14]. Abstract features denote features not mapped to concrete artefacts. Additionally, feature models support constraints for the selection of features in different branches and allow an arbitrary model depth. But for the sake of readability and ease of understanding, the depths of individual branches should be as uniform as possible [Ba05].

Finally, we compared the surveyed approaches by analysing the feature models, and synthesized a consolidated feature model to uniformly describe message queuing systems in the context of industrial production, see Section 3. For feature modeling, we used FeatureIDE [Me17], as shown in Fig. 4.[6]

2.3 Survey Results

In this section, we present the results of the feature model analysis of the surveyed models and architectures. For the sake of brevity, the section represents a summary of

[6] See https://featureide.github.io/

the analysis.

Systems: In his book, Stopford specifically discusses the application of the open source event streaming platform Apache Kafka to the problem of tying back shared facts in distributed business information systems to a "single thread of irrevocable truth" [St18]. To do so, the author introduce streaming and Kafka, discuss patterns and techniques for development, and present a system example using Kafka Streams and KSQL. Shapira et al. also provide an extensive discussion of Apache Kafka [Sh21], beginning with an overview of typical use cases for Kafka in a data ecosystem. Based on Apache Kafka, Confluent [Co20] provides an overview of the architecture of the Confluent Platform. The white paper is intended to serve as a reference and guideline for data architects and system administrators who are planning to deploy the platform in production.

Apache Pulsar is a distributed publish-subscribe messaging system originally created at Yahoo and provided by the Apache Software Foundation [Ta23]. In his Master's thesis, Nuikka considers it a significant competitor to Apache Kafka for its performance characteristics, but notes that Pulsar lacks both community support and technological maturity [Nu21].

Madhu and Sunanda [MS19] give a brief overview of the main features of the RabbitMQ messaging system and discuss its use of the Advances Message Queuing Protocol (AMQP). They also present the different types of exchange used to propagate messages to queues.

Theorin et al. present an event-driven service oriented architecture named Line Information System Architecture (LISA) [Th15]. LISA consists of a message bus, a custom message exchange format, and communication and service endpoints. As a message bus, LISA uses the middleware ActiveMQ. Communication endpoints represent low-level adapters between devices and the message bus, and service endpoints manage transformations of low-level events to standardized structures for connectivity to manufacturing execution systems (MES) and calculating key performance indicators (KPIs). Fill transformations append a set of static attribute-value pairs to a given event, Map transformations append a set of attribute-value pairs based on the current state of the system, and Fold transformations aggregate a finite sequence of events into a single event. In their paper, the authors emphasize the need for retrofitting devices in the production plant as they are based on different technologies and come from different eras.

Cybus Connectware is a commercial on-premise data integration platform specially developed for industrial production at the shop floor [Cy22]. It integrates a variety of shop floor data sources, protocols and formats such as Modbus, Message Queuing Telemetry Transport (MQTT), MTConnect, OPC UA and Profinet, thus providing a scalable factory-wide data inventory. At its core, its architecture consists of a MQTT broker. Connectware supports several data consumers including Cloud providers (e.g., Microsoft Azure and Amazon Web Services), IoT Applications (e.g., Siemens Mindsphere), data analytics applications (Grafana, Tableau and Elasticsearch) and local

IT systems such as SAP ERP and other enterprise resource planning (ERP) and manufacturing execution systems (MES). Cybus connectivity strategy is based on changeless machine connection that aims to avoid configuration changes to Programmable Logic Controllers (PLCs).

ZeroMQ [Ze23] is an asynchronous programming library that uses Sockets for high-throughput distributed systems. Although based on the message queuing paradigm, it does not require a dedicated message broker but can be used to implement a broker-based message exchange pattern. In his thesis [Pf22], Pfefferkorn finds evidence that ZeroMQ is suitable as a communication basis for distributed control and automation systems (DCS). To do so, he implemented the individual system nodes using Raspberry Pi single-board computers, on which the IEC 61499 standard for distributed automation is supported using the open source framework 4DIAC.

Models: Bruns and Dunkel provide an in-depth discussion of Event-Driven Architecture (EDA) in general, and Complex Event Processing (CEP) in particular [BD10]. The topics are discussed from three different perspectives, the technical benefits for companies, software design, and practical implementation. The EDA introduced by Bruns and Dunkel can be regarded as a high-level reference architecture. Additionally, depending on the use case, they recommend modifications for sensor networks and analysis systems.

Fu et al. [Fu21] compare five popular MQ systems that are representative in multiple aspects. The work includes a summary of the main features of MQ systems in general. These are discussed in terms of system production, quality of service, and performance.

In [So18], Sommer et al. present a comparison of four well-established message-oriented middleware systems in terms of qualitative and quantitative criteria, with the focus on performance and their suitability for industrial production.

Yongguo et al. [Yo19] discuss the main features of message-oriented middleware (MOM) based on a literature research and compares the main features and application scenarios of several common MOM products, namely JMS, Kafka, ZeroMQ, MQTT, AMQP, and RocketMQ. The table presented by Yongguo et al. that summarizes their comparison can be regarded as a simple feature hierarchy for MOM.

Dobbelaere and Esmaili [DE17] conduct a qualitative and quantitative/empirical comparison of the two popular and commercially supported open source pub/sub systems Apache Kafka and RabbitMQ. To do so, they establish a common comparison framework based on the core functionalities of pub/sub systems, highlight the distinct features of and best-suited use cases for the two systems. They conclude their survey with a decision table for the two systems.

Mishra and Kertesz present an extensive survey on the use of the Message Queuing Telemetry Transport (MQTT) protocol for machine-to-machine (M2M) communication and Internet of Things (IoT) systems [MK20]. They also propose a taxonomy to compare the properties of various concrete MQTT broker systems and programming libraries to

make it easier for scientists and end-users to select a specific MQTT implementation.

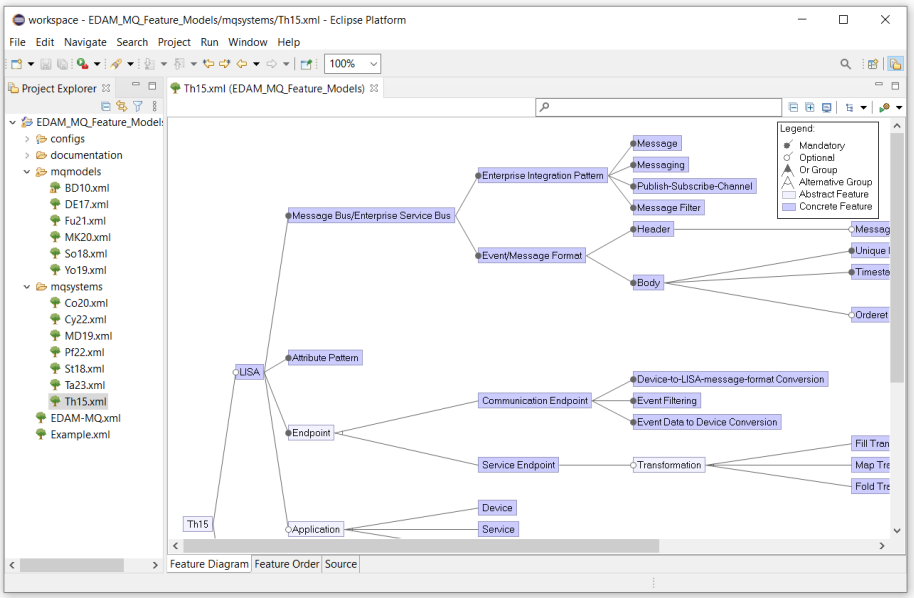


Fig. 4: Modeling with FeatureIDE

Summary: Overall, the work differs in terms of abstraction, as discussed above and shown in Tab. 1. The level of abstraction considers whether an approach describes a concrete architecture (system) or a generic representation or discussion of architectures (model). In addition, domain orientation considers if an approach is oriented towards an application domain or not (i.e., generic vs. manufacturing). Out of the 14 approaches, 5 are oriented towards manufacturing. We also consider 5 of the work as stemming from industry, while most of the work are of academic origin.

Formality shows how the authors describe their respected approach (e.g., using a formal taxonomy or model language, block diagrams, informal lists and/or plain text). We regard all of the work considered as informal (text, tables and diagrams), which limits its comparability and allows for unwanted interpretations. Thus, in the following section, we provide an aggregated feature model to fill the lack of a formal and understandable model for event-driven broker architectures in manufacturing. However, although the work considered is relatively recent, the number is small and can therefore only be considered a good start to more rigorous research.

Work	Abstraction	Domain	Type of Work	Formality	Provenance
[BD10]	model	generic	book	informal	academia
[Fu21]	model	generic	paper	informal	academia

[So18]	model	manufacturing	paper	informal	academia
[Yo19]	model	generic	paper	informal	academia
[DE17]	model, system	generic	paper	informal	academia
[MK20]	model	manufacturing	paper	informal	academia
[St18]	system	generic	book	informal	industry
[Sh21]	system	generic	book	informal	industry
[Co20]	system	generic	whitepaper	informal	industry
[Ta23]	system	generic	online	informal	industry
[MS19]	system	generic	paper	informal[7]	academia
[Th15]	system	manufacturing	paper	informal	academia
[Cy22]	system	manufacturing	whitepaper	informal	industry
[Pf22]	system	manufacturing	thesis	informal	academia

Tab. 1: Overview of survey results

3 Features of Event-Driven MQ Architectures in Manufacturing

In principle, event-based or event-driven architectures (EDA) use information on events in order to communicate between services that are decoupled from one another. Such architectures are usually push-oriented, with all events processed only when needed [BD10]. If middleware systems based on the principle of message queuing (message-oriented middleware, MOM) are used as integration platforms, various criteria can be considered, see [Fu21], [Yo19]. Based on the analysis of the surveyed approaches in Subsection 2.3, we developed a consolidated feature model to describe and evaluate *event-driven message queuing architectures in manufacturing* (EDAM-MQ).

It is important to note that, as a model for comparison, a feature of the EDAM-MQ represents the description of a part or aspect of a concrete system, not the system aspect itself. Therefore, the selection/deselection of a feature represents the presence or absence of the respective description. The feature model consists of four main features: use case features, architecture-related features, system realization features, and quality of service-related features, see Fig. 5.

3.1 Use Case Features

Regarding the use cases of an event-driven message queuing architecture, we adopted the general use cases discussed by Shapira et al. [Sh21], namely activity tracking, *user notification*, *metrics logging*, *commit logging* and *stream processing*. We also added the specific use case of *industrial edge device connectivity* [Cy22], as it represents a fundamental challenge to manufacturing that employs a variety of devices "based on different technologies from different eras" [Th15]. Fig. 5 shows the features for use case descriptions.

[7] However, the Advanced Message Queuing Protocol (AMQP) discussed in [MD19] is approved as an international Standard, see [Oa14].

Fig. 5: Main features of the EDAM-MQ feature model; third-level sub-features collapsed

3.2 Architecture-related Features

Architecture-related features provide an overview of the architecture of an event-driven message queuing system. We consider *system deployment*, *cluster management*, *system components*, the native *application protocols*, *client types*, and the *message queuing model* as principal sub-features, see Fig. 6. First, system deployment can occur *cloud-hosted* or *on-premise* [Co20], and descriptions of cluster management includes mechanisms for *load balancing* [Yo19], *failure prevention* [Yo19] [St18], *local network server clustering* [MD19] and *replication* [St18].

Typical system components include *brokers*, a *management user interface* (UI), and storage systems for the overall system *configuration*, for *metadata*, and for *queues/messages*. Regarding the brokers, either a *master-slave* or a *peer-to-peer* operation model is usually applied [Fu21]. Moreover, regarding the Management UI, typical features include *status monitoring* [Yo19], *system configuration* [Yo19], *user management* [MD19], *permission management* [MD19], *queue management* [MD19] and *message exchange management* [MD19]. Thereby, status monitoring can provide *machine-centred* or *order-centred views* [Th15].

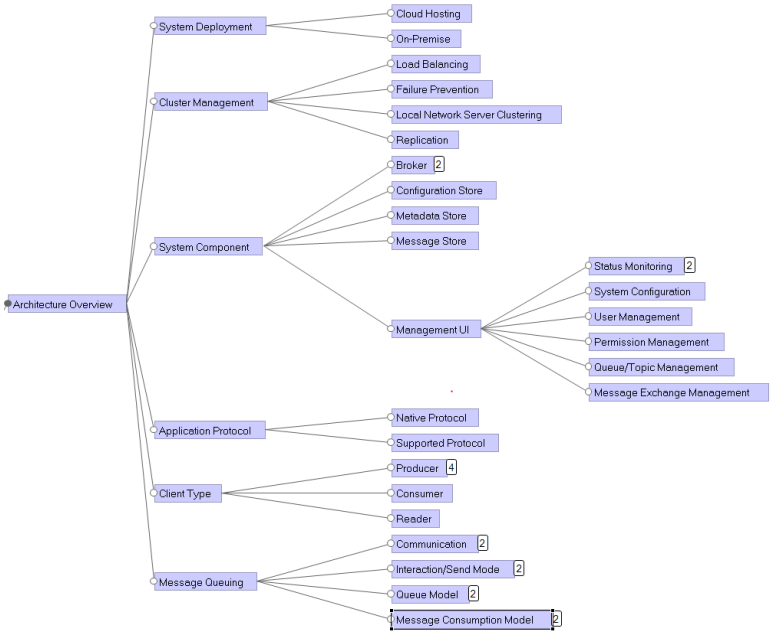


Fig. 6: Architecture-related description features

Message queuing systems usually apply a specific application protocol for messaging *natively*, and *support* several protocols additionally. Typical protocols include AMQP, STOMP, MQTT, XMPP, and HTTP [Yo19] [Fu21] [DE17]. Clients of such systems are distinguished in *producers* and *consumers*. In addition, we also consider *readers* that read but do not consume messages [Ta23]. Producers can *share access* to message queues; can have *exclusive access*, *exclusive access with fencing* or *wait for exclusive access* [Ta23].

Message queuing is discussed in terms of *communication*, *interaction*, the *queue model* applied, and the *message consumption* model. The communication model distinguishes between *point-to-point* communication and *publish-subscribe* communication [Yo19] [Fu21] [So18], while interaction mode differs between *request-reply* and *fire-and-forget* client interaction [Yo19]. Queue models can be *hierarchical* [Cy22], such as supported by MQTT, or *non-hierarchical*. Finally, message queuing either employs *push*- or *pull*-based message consumption [Fu21].

3.3 System Realization Features

Regarding the realization of the system under study, we consider the feature descriptions for *system compatibility*, *device management*, *licensing*, and *implementation*. Fig. 7 presents an overview of these features.

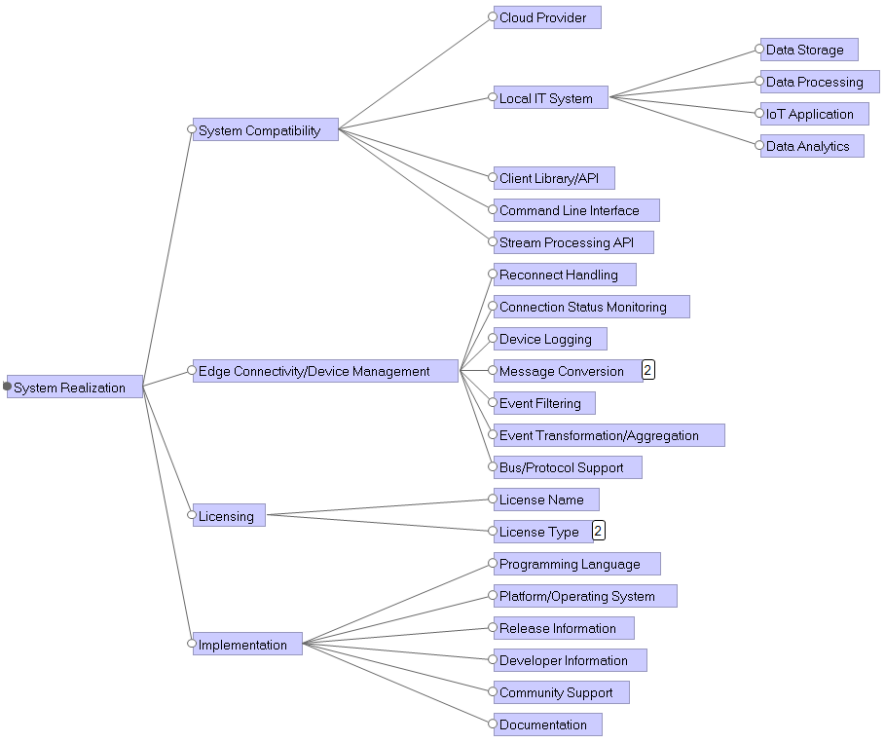


Fig. 7: System realization features

System compatibility [Cy22] can discuss *compatible cloud providers* such as Microsoft Azure and Amazon Web Services (AWS), a variety of *local IT systems* such as SAP ERP [Cy22] and HDFS [Fu21], *client libraries* and *application programming interfaces* (APIs), *command line interfaces* (CLIs) and *stream processing APIs* such as Kafka Streams and KSQL [St18].

Device management [Cy22], on the other hand, has to guarantee connectivity to a wide variety of Edge devices [Th15]. Typical features include *reconnect handling* [Cy22], *connection status monitoring* [Cy22], *device logging* [Cy22], *message conversion* (*message-to-device* and *device-to-message*) [Th15], and *event filtering* and *event transformation/aggregation* [Th15]. To do so, device management must support a variety of *bus systems/protocols* such as Modbus, OPC UA and Profinet [La21] [Cy22].

Licensing discusses the *name* and the *type* (*closed* or *open source*) of the system's software license [MK20] [So18]. Implementation names the primary *programming language* with which the system was realized and specifies the required *platform/operating system* [Fu21] [MK20] [MD19] [So18]. In addition, we consider information on the *software release*, the *developer*, on *community support* and on *documentation* as relevant for comparison [MK20] [So18].

3.4 Quality of Service-related Features

For the long-term and sustainable operation of a system, mechanisms to ensure service quality are of particular importance. Regarding event-driven message queuing systems, we consider guarantees for *message delivery* and *message ordering*, and the mechanisms to promote *security*, *performance*, and *reliability* as paramount. Quality of Service-related features are shown in Fig. 8

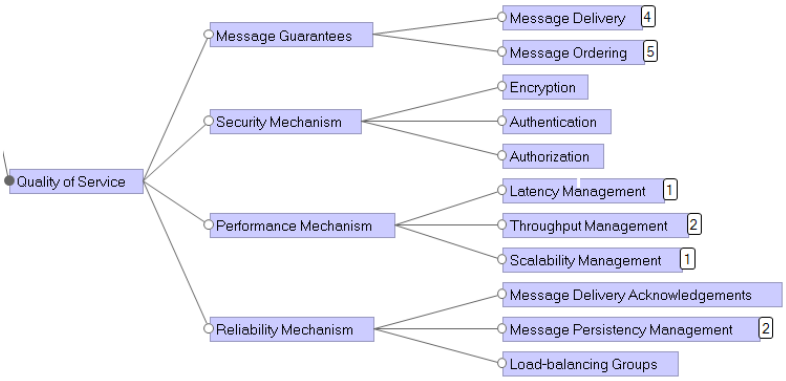


Fig. 8: Quality of Service-related feature descriptions

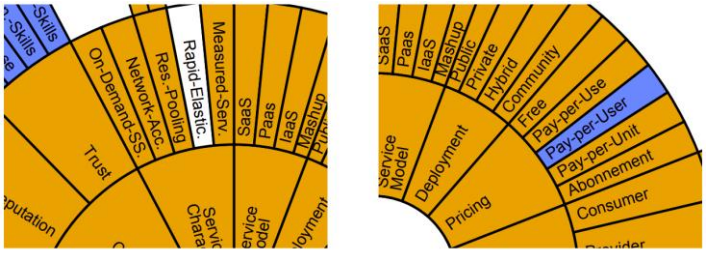
Message delivery typically considers the guarantees *at-most-once* (interval [0..1]), *at-least-once* (interval [1..n]), and *exactly-once* (interval [1..1]) [Fu21] [DE17]. If none is supported, a guarantee of *maybe* (interval [0..n]) is assumed.[8] Message ordering can guarantee *partition-/queue-ordering*, *global ordering* of messages, or *no ordering* at all [Fu21] [Yo19] [DE17].

Regarding security, we include mechanisms for *encryption*, *authentication* and *authorization*. Encryption is often provided by SSL/TLS [Yo19]; authentication includes SASL [Yo19] [So18] and x.509 certificates [Cy22], amongst others. Authorization can be provided by access control list (ACL) [St18] [So18] and role-based access control (RBAC) [Co20]. Performance-related features include *latency management*, *throughput management* and *scalability management*. Latency can be addressed by optimizing *memory access* [Fu21], throughput by *message batching* [Fu21] [St18] and *partitioning* [St18] and scalability by using *quotas* [St18], for example. Finally, reliability mechanisms include *message delivery acknowledgements* [MD19], *message persistence management* (*in-memory* or *on-disk*) [MD19] and *load-balancing groups* [St18].

[8] Note that the message delivery guarantee of *at-least-once* requires message persistence and that *exactly-once* requires transactional behavior for consuming and processing a message.

4 Future Work

Regarding the description of concrete event-driven systems for manufacturing that are based on the message queuing paradigm, the description of a feature according to the model presented in Section 3 can be said to be present or not. Gudenkauf et al. describe a concept for the visual representation of the state of a feature model instance using radial space-filling sunburst diagrams, coloring and interactive visualization using cloud services as an example [Gu13]. Their so-called Cloud Service Navigators (CSN) also support additional user-defined attributes, as shown in Fig. 9 (b). Adapting the CSN concept to the field of event-driven manufacturing can help companies improve long-term viable decisions about mission-critical middleware. We also advise a more rigorous literature review to increase the validity of the model, see Section 2.3.



(a) Absent service feature. (b) User-defined service feature.

Fig. 9: (a) Section of a CSN that does not describe the feature „Rapid-Elasticity“; (b) Section of a CSN that extends the “pricing” feature with a user-defined “pay-per-user” pricing model

In Addition to the features of event-driven architectures, scenario-based methods have proven themselves for the qualitative evaluation of architectures, since they also take into account the risks of design decisions [Gu23]. Gudenkauf et al. describe an interactive approach and proper tooling to architecture evaluation based on the well-known Architecture Tradeoff Analysis Method (ATAM). This approach, Interactive Software Architecture Analysis (ISAA) can possibly be combined with CSN's interactive feature modeling to form a holistic decision system.

5 Conclusion

Distributed applications based on the principle of message queuing (MQ) and event handling are attracting increasing interest in the manufacturing industry. However, the selection of a concrete system architecture represents a significant decision with long-term consequences for a company. To improve the overview of the features of such systems, we surveyed different recent models and architectures for event-driven MQ systems. Based on this survey, we propose a consolidated feature model EDAM-MQ to uniformly describe and evaluate event-driven manufacturing systems, regardless of

whether an organization's own architectural needs or the offers of external providers are evaluated. As future work, we also envision the integration of interactive feature modeling for EAM-MQ systems and interactive architecture analysis based on quality scenarios as a promising experiment to form a holistic decision system.

Bibliography

[Ba94] Basili, V.; Caldiera, G.; Rombach, H. D.: Goal Question Metric Approach: Encyclopedia of Software Engineering. John Wiley & Sons, Inc., pp. 528-532, 1994.

[Ba05] Batory, D.: Feature Models, Grammars, and Propositional Formulas. In (Hutchison, D. et al. Eds.): Software Product Lines. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 7–20, 2005.

[BD10] Bruns, R.; Dunkel, J.: Event-driven architecture. Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse. Springer, Berlin, Heidelberg, 2010.

[Bo08] Borthakur, D.: HDFS Architecture Guide. Hadoop apache project 53, 2008.

[CE00] Czarnecki, K.; Eisenecker, U. W.: Generative programming. Methods, tools, and applications. Addison-Wesley, Boston, 2000.

[Ce21] Cerquitelli, T. et al.: Manufacturing as a Data-Driven Practice: Methodologies, Technologies, and Tools. Proceedings of the IEEE 4/109, pp. 399–422, 2021.

[CH06] Czarnecki, K.; Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 3/45, pp. 621–645, 2006.

[Co20] Confluent, Inc.: Confluent Platform Reference Architecture, 2020.

[Cy22] Cybus: Cybus Connectware Technical Specification. https://www.cybus.io/wp-content/uploads/2022/02/cybus-connectware-technical-specifications-siemens-industrial-edge.pdf, accessed 22 May 2023.

[DE17] Dobbelaere, P.; Esmaili, K. S.: Kafka versus RabbitMQ. The first public release of this report appeared in: P. Dobbelaere and K. S. Esmaili. Kafka versus RabbitMQ: A comparative Study of two Industry Reference Publish/Subscribe Implementations. In Proceedings ofthe 11th ACM International Conference on Distributed and Event-based Systems, pages 227–238. ACM, 2017. http://arxiv.org/pdf/1709.00333v1.

[Di21] DIN EN IEC 62541-4:2021-08:2021-08-01, OPC Unified Architecture – Teil 4: Dienste (IEC 62541-4:2020); Englische Fassung EN IEC 62541-4:2020.

[Ev04] Evans, E.: Domain-driven design. Tackling complexity in the heart of software. Addison-Wesley, Boston, Munich, 2004.

[Fr21] Frochte, J.: Maschinelles Lernen. Grundlagen und Algorithmen in Python. Hanser, München, 2021.

[Fu21] Fu, G.; Zhang, Y.; Yu, G.: A Fair Comparison of Message Queuing Systems. IEEE Access 9, pp. 421–432, 2021.

[Gu13] Gudenkauf, S. et al.: A Reference Architecture for Cloud Service Offers. In (Gašević, D.

et al. Eds.): Proceedings of the 17th IEEE International EDOC Conference. IEEE Computer Press, Vancouver, Canada, pp. 227–236, 2013.

[Gu23] Gudenkauf, S.; Bachmann, U.; Hartmann, N.: A Concept and a Multitenant Web Application for Interactive Software Architecture Analysis. In (Sales, T. P. et al. Eds.): Enterprise Design, Operations, and Computing. EDOC 2022 Workshops. Springer International Publishing, Cham, pp. 268–283, 2023.

[Jo21] Joseph, J.: Mastering Apache Pulsar. O'Reilly Media, Inc, 2021.

[KB13] Kitchenham, B.; Brereton, P.: A systematic review of systematic review process research in software engineering. Information and Software Technology 12/55, pp. 2049–2075, 2013.

[Ki07] Kitchenham, B. et al.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Version 2.3. EBSE Technical Report, 2007.

[Kl17] Kleppmann, M.: Designing data-intensive applications. The big ideas behind reliable, scalable, and maintainable systems. O'Reilley, Beijing, Boston, Farnham, Sebastopol, Tokyo, 2017.

[Kr11] Kreps, J.; Narkhede, N.; Rao, J.: Kafka: a Distributed Messaging System for Log Processing: NetDB'11, 2011.

[La21] Langmann, R.: Vernetzte Systeme für die Automatisierung 4.0. Bussysteme - Industrial Ethernet - Mobile Kommunikation - Cyber-Physical Systems. Hanser, München, 2021.

[LF22] Lewis, J.; Fowler, M.: Microservices. a definition of this new architectural term. https://martinfowler.com/articles/microservices.html, accessed 14 Jun 2022.

[LM10] Lakshman, A.; Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review 2/44, pp. 35–40, 2010.

[Me17] Meinicke, J. et al.: Mastering software variability with FeatureIDE. Springer, Cham, 2017.

[Mi18] Mishra, B.: Performance Evaluation of MQTT Broker Servers. In (Gervasi, O. et al. Eds.): Computational Science and Its Applications – ICCSA 2018. Springer International Publishing, Cham, pp. 599–609, 2018.

[MK20] Mishra, B.; Kertesz, A.: The Use of MQTT in M2M and IoT Systems: A Survey. IEEE Access 8, pp. 201071–201086, 2020.

[MS19] Madhu, M. P.; Sunanda, D.: Distributing Messages Using Rabbitmq with Advanced Message Exchanges. International Journal of Research Studies in Computer Science and Engineering 2/6, 2019.

[Nu21] Nuikka, J.: Comparison of Cloud Native messaging technologies. Master's thesis. Tampere University, 2021.

[Oa14] OASIS Advanced Message Queuing Protocol (AMQP) TC: ISO/IEC 19464:2014(E). Information technology — Advanced Message Queuing Protocol (AMQP) v1.0 specification. ISO copyright office, Switzerland, 2014.

[Pa08] Palankar, M. R. et al.: Amazon S3 for science grids. In (Kosar, T. Ed.): Proceedings of the 2008 international workshop on Data-aware distributed computing - DADC '08.

ACM Press, New York, New York, USA, pp. 55–64, 2008.

[Pf22] Pfefferkorn, D.: Verwendung von ZeroMQ in verteilter Automatisierung, Vorarlberg, 2022.

[Ra19] Ramasamy, K.: Unifying Messaging, Queuing, Streaming and Light Weight Compute for Online Event Processing: Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems. ACM, New York, NY, USA, p. 5, 2019.

[SG16] Siepmann, D.; Graef, N.: Industrie 4.0 – Grundlagen und Gesamtzusammenhang. In (Roth, A. Ed.): Einführung und Umsetzung von Industrie 4.0. Grundlagen, Vorgehensmodell und Use Cases aus der Praxis. Springer Gabler, Berlin, Heidelberg, pp. 17–82, 2016.

[Sh22] Shapira, G. et al.: Kafka. The Definitive Guide Real-Time Data and Stream Processing at Scale. O'Reilly, Sebastopol CA, 2022.

[SJ14] Shahrivari, S.; Jalili, S.: Beyond Batch Processing: Towards Real-Time and Streaming Big Data. arXiv, 2014.

[So18] Sommer, P. et al.: Message-oriented Middleware for Industrial Production Systems: 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE). IEEE, pp. 1217–1223, 2018.

[St18] Stopford, B.: Designing Event-Driven Systems. Concepts and Patterns for Streaming Services with Apache Kafka. O'Reilly, Sebastopol, 2018.

[Ta23] The Apache Software Foundation: Apache Pulsar Docs Version 3.0.x. Concepts and Architecture. https://pulsar.apache.org/docs/3.0.x/concepts-overview/, accessed 28 May 2023.

[Th15] Theorin, A. et al.: An Event-Driven Manufacturing Information System Architecture. IFAC-PapersOnLine 3/48, pp. 547–554, 2015.

[Th14] Thüm, T. et al.: FeatureIDE: An extensible framework for feature-oriented software development. Science of Computer Programming 79, pp. 70–85, 2014.

[Wo18] Wolff, E.: Microservices. Grundlagen flexibler Softwarearchitekturen. dpunkt.verlag, Heidelberg, 2018.

[Yo19] Yongguo, Jiang and Qiang, Liu and Changshuai, Qin and Jian, Su and Qianqian, Liu: Message-oriented Middleware: A Review: 2019 5th International Conference on Big Data Computing and Communications (BIGCOM). IEEE, pp. 88–97, 2019.

[Ze23] The ZeroMQ authors: ZeroMQ. An open-source universal messaging library. https://zeromq.org/, accessed 22 May 2023.